OULUN YLIOPISTO

TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

ELEKTRONIIKAN JA TIETOLIIKENNETEKNIIKAN TUTKINTO-OHJELMA

KANDIDAATINTYÖ

KONENÄÖN KÄYTTÖ GRAAFISEN KÄYTTÖLIITTYMÄN TESTIAUTOMAATIOSSA

Tekijä Oskar Byman

Ohjaaja Juha Häkkinen

Kesäkuu 2021

Byman O. (2021) Konenäön käyttö graafisen käyttöliittymän testiautomaatiossa. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 25 s.

TIIVISTELMÄ

Työssä sovelletaan konenäön menetelmiä sulautetun järjestelmän graafisen käyttöliittymän testiautomaatiossa. Rakennetaan siis järjestelmä, joka tunnistaa laitteen tilan ja osaa liikkua tilojen välillä näppäinpainallusten avulla. Järjestelmässä on kolme tärkeää osa-aluetta: tilojen välillä liikkumisen mahdollistava virtuaalinen näppäimistö, tilakartta ja konenäköä soveltava testiautomaatiokehikko. Työn teoriaosuudessa käsitellään ja vertaillaan eri menetelmiä osakuvien etsintään kokonaiskuvasta. Vertailtavat menetelmät löytyvät OpenCV kirjastosta. Työ tehtiin Bittium Oyj:n projektin tarpeeseen. Työssä tuotettiin konenäköä käyttävä graafisen käyttöliittymän testikirjasto, jonka pohjalta luotiin testisekvenssejä. Kirjasto sisältää virtuaalisen näppäimistön käytön, kuvien kaappauksen, tilan seurannan ja kuvien tarkasteluun käytetyt metodit. Työ testattiin toimivaksi ja otettiin käyttöön projektin testiautomaatioon.

Avainsanat: OpenCV, konenäkö, testiautomaatio

Byman O. (2021) Computer vision in embedded systems graphical user interface testing. University of Oulu, Degree Programme in Electronics and Communications Engineering. Bachelors's Thesis, 25 p.

ABSTRACT

This thesis covers the usage of Computer Vision in embedded systems graphical user interface test automation. The goal is to build a system that can recognize the systems state and move between states by pressing virtual buttons. The built system can be divided into three main parts. The parts are a virtual keypad, a state map and the test automation frame. The theoretical part of the thesis covers different template matching methods, which can be found in the OpenCV library. This thesis was requested by Bittium Oyj. The product of this project was a library for the test automation framework. This library contained the virtual keypad functionality, screen capturing, state tracking and the computer vision methods. The project was tested and taken into use in test automation.

Key words: Computer Vision, OpenCV, Test automation.

SISÄLLYSLUETTELO

ALKULAUSE

Tämä työ tehtiin Bittium Oyj:lle. Haluaisin kiittää ohjaajiani Yliopistotutkija Juha Häkkistä ja Bittiumin edustajaa Aki Partasta ohjauksesta työn rakentumisen eri vaiheissa. Työ sai alkunsa projektin tarpeesta kehittää graafisen käyttöliittymän testausmahdollisuuksia, josta tehtiin suunnitelma työn toteutukselle.

Oulussa 23.6.2021

Oskar Byman

LYHENTEIDEN JA MERKKIEN SELITYKSET

CV	Computer Vision, eli konenäkö
JSON	JavaScript Object Notation, tiedostomuoto
PNG	Portable Network Graphic, kuvatiedostomuoto
RGB	Red Green Blue, Värillisen pikselin muodostavat kolme väriä
TM	Template Matching, osakuvan etsintä jostain suuremmasta kuvasta
OCR	Optical Character Recognition, optinen merkin tunnistus
R	Result eli tulos
T	Template eli osakuva
I	Image eli kuva
w	Leveys
h	Korkeus

1. JOHDANTO

Konenäkö on nykypäivänä jo vallitseva, mutta myös kasvava aihe tekniikan alalla. Sitä sovelletaan jo lähes kaikkialla laajasti monissa laitteissa, kuten tuotantoketjujen valvonnassa tai sormenjälkitunnistimessa. Konenäkö oikein käytettynä voi tehostaa ja automatisoida monia prosesseja, jotka eivät ennen olleet mahdollisia ilman ihmissilmän valvontaa.

Tämän työn tarkoituksena on tuottaa Bittium Oyj:lle, sulautetun järjestelmän graafisen käyttöliittymän testaukseen konenäköön perustuva kirjasto, jota voidaan soveltaa jatkossakin automaatiotestauksen kehityksessä. Graafisen käyttöliittymän tehokas testaaminen on vaikeaa ilman, että käyttöliittymää voi nähdä, jolloin konenäön avulla käyttöliittymän testaamista voidaan tehostaa ja laajentaa.

Työn teoreettisessa osiossa käydään läpi konenäön historiaa ja työssä käytettyä teoriaa. Sovellettu teoria liittyy Template Matching-käsitteeseen. Teknisessä osassa tuotetaan ohjausosa sulautetulle järjestelmälle ja sovelletaan OpenCV kirjastoa järjestelmän eri tilojen tunnistamiseen, niiden väliseen navigointiin ja osakuvan etsintään.

Tavoitteena työlle on tuottaa esimerkkejä automatisoiduista testeistä, joissa käydään jokin sekvenssi järjestelmän eri tilojen läpi ja tutkitaan tilojen oikeellisuutta ja etsitään niistä osakuvia.

Tutkimuskysymykset teoriaosuutta varten:
- Millainen konenäön matemaattinen perusta on?
- Mitä eri algoritmeja käytetään kuvien vertailuun?

Tekniset tavoitteet työlle, työn tulisi:
- Pystyä navigoimaan ympäri käyttöliittymää määritetyn kartan mukaan.
- Pystyä vertailemaan nykyistä tilaansa odotettuun tilaan.
- Raportoida tuloksensa logituksen muodossa.
- Pystyä upottamaan osaksi muuta testiautomaatiota, jolloin sen päivittäinen käyttö olisi mahdollista.

2. KONENÄKÖ

Konenäkö eli tietokoneen kyky tulkita visuaalista informaatiota, koettaa jäljitellä ihmisen kykyä erotella visuaalisia muotoja, värejä ja kappaleita. Konenäön keskeisiä ongelmia on sen käänteinen luonne. Tavallisesti ongelmasta saadaan informaatiota ja ominaisuuksia, joiden perusteella tuotetaan jokin tulos, mutta konenäön tapauksessa saadaan kuva, josta täytyy saada irti yleismuotoisia ominaisuuksia ja informaatiota, jotka auttavat tunnistamaan vastaavanlaisia kuvia jatkossakin. [1]

Konenäön haasteista huolimatta sille on kyetty tuottamaan useita menetelmiä ja algoritmeja, joita on myöhemmin kyetty soveltamaan useisiin eri käyttötarkoituksiin. Esimerkkejä käyttötarkoituksista ovat optinen hahmontunnistus, 3D kuvantaminen ja suunnittelu, liikkeentunnistus ja biometriset sovellukset, kuten optiset sormenjälkitunnistimet. [1] Hieman jo ikääntynyt, mutta silti kattava lista konenäön sovellutuksista löytyy David Lowen listauksesta, The Computer Vision Industry [1, 2]

2.1. Konenäön historiaa

Konenäön pitkä historia alkoi noin 50 vuotta sitten aikaisella 1970-luvulla. Tähän aikaan kuviteltiin, että visuaalisen datan käsittely oli vain pieni askel kohti suurempien ja vaikeamman oloisten ongelmien ratkaisua. 70-luvulla konenäköä aloitettiin etsimällä kuvista kappaleiden välisiä reunoja ja koittamalla rekonstruoida niistä 3D-malleja. [1]

80-luvulle tultaessa huomio siirtyi kehittyneiden matemaattisten menetelmien soveltamiseen konenäön käyttötarkoituksissa. Stereonäön käyttöä alettiin hyödyntämään 3D-mallien luontiin ja reunan tunnistamisen algoritmeja paranneltiin. [1]

90-luvulla konenäön kehitys jatkui samojen teemojen parissa, mutta monet osa-alueet saivat enemmän erityistä huomiota konenäön kasvaessa käsitteenä. Muun muassa stereonäkö algoritmeja paranneltiin käyttämään useampia lähteitä, jolloin suurempi tarkkuus 3D-mallien muodostamiseen saavutettiin. Videoiden liikkeenseuranta algoritmit kehittyivät monien tahojen ansiosta. Uutena osa-alueena tilastolliset oppimismenetelmät aloittivat toiminnan kasvojentunnistuksessa. 90-luvun tärkeimpiä uusia mullistuksia oli silti kuvien muuntelu, jonka avulla valmiista kuvista pystyttiin ottamaan osa-alueita ja muotoilemaan niitä animaation omaisesti. [1]

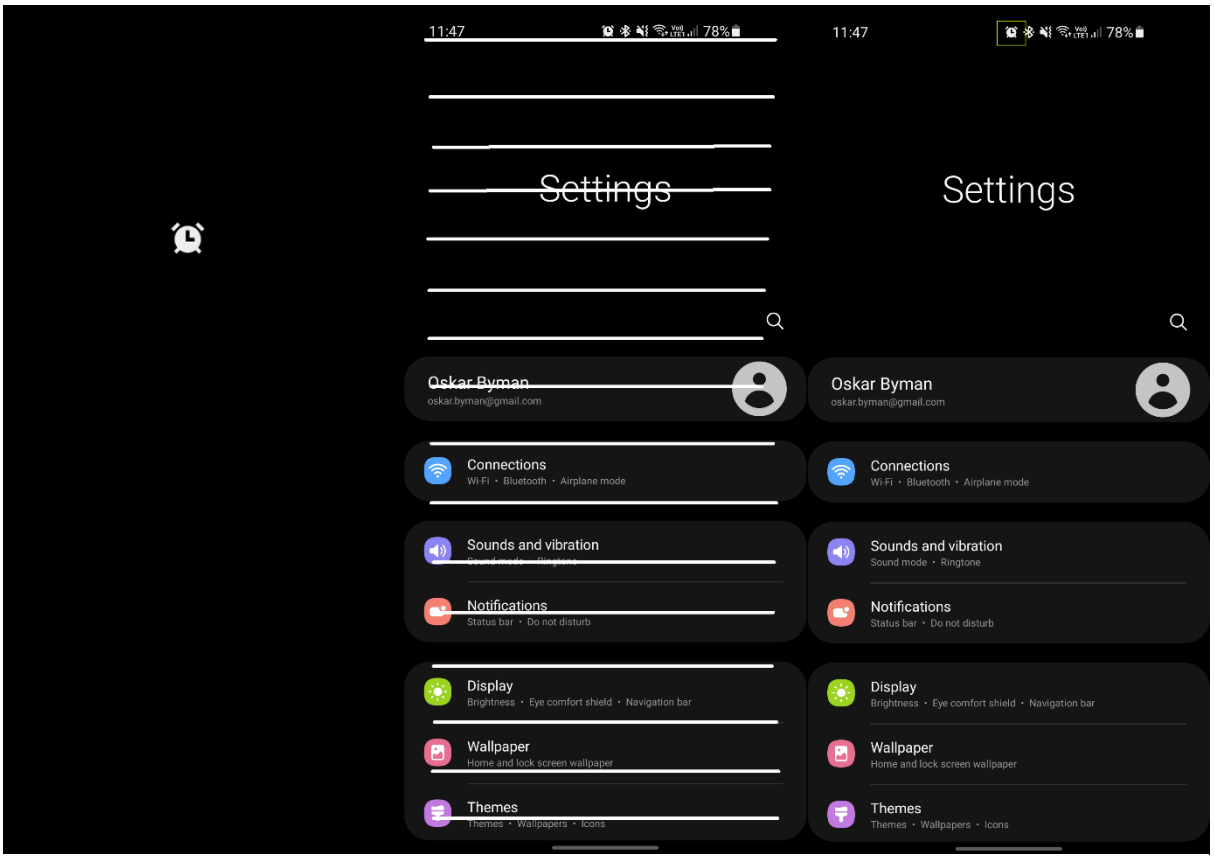
2000-luvulle saavuttaessa kaikki 90-luvun teemat kehittyivät. Kompleksisia algoritmeja optimoitiin ja kuvan syntetisointia muista kuvista paranneltiin. Tärkeimpänä teemana kuitenkin oli koneoppimisen soveltaminen konenäön ongelmiin, jota vauhditti internetistä löytyvä valtava määrä dataa koneoppimisen tueksi. [1]

2.2. Konenäön algoritmit

Konenäkö kattaa useita algoritmeja, jotka kehittyvät jatkuvasti. Tässä työssä käsitellään ainoastaan template matching algoritmeja ja käytettäviä menetelmiä. Muita algoritmeja ja sovelluksia löytyy Szeliskin Konenäköä käsittelevästä kirjasta. [1]

Template matching on karkeasti suomennettuna osakuvan etsintää. Siinnä otetaan jokin alkuperäistä tutkittavaa kuvaa pienempi kuva ja sitä vastaavia osioita etsitään alkuperäisestä kuvasta. Template matchingissä verrataan osakuvaa alkuperäisen kuvan jokaiseen kohtaan eri menetelmillä. Näitä menetelmiä ovat muun muassa normalisoitu ristikorrelaatio, korrelaatiokerroin ja neliöllinen erotus. Nämä kaikki kolme ja niiden normalisoidut versiot löytyvät OpenCV kirjastosta. Jokaisessa menetelmässä kuvaa siirretään pikseli kerrallaan kuvan yli ja suoritetaan vertailu joka kohdassa. Alkuperäiseen kuvaan lisätään reunoille nollia,

jotta osakuvaa voidaan verrata valitulla menetelmällä jokaiseen kohtaan alkuperäisessä kuvassa, keskikohdan osuessa jokaiseen alkuperäiseen pikseliin. Tämän ansiosta vertailun tulosmatriisi on samankokoinen kuin alkuperäinen kuva. Tulosmatriisin avulla voidaan päätellä missä kohdassa osakuva ja alkuperäinen kuva olivat samanlaisia. [4]



Kuva 1. Template Matchingin vaiheet: Valitaan etsittävä osakuva, vertaillaan sitä joka kohdassa alkuperäisessä kuvassa ja merkataan vertailun kynnysarvon ylittäneet kohdat.

Kuvassa 1 vasemmalla näkyvä herätyskelloikoni etsitään kokonaiskuvasta liu'uttamalla sitä tarkasteltavan kuvan ylitse ja lukemalla korrelaatiokertoimen arvo jokaisessa kohdassa. Tuloksena saadusta tulosmatriisista luetaan suurimmat arvot, jotka suodatetaan halutun kynnysarvon perusteella. Herätyskello löytyy kuvan oikeasta ylänurkasta ilmoitusriviltä. Jos kuvasta olisi haettu kuvaa, jossa esiintyy numero seitsemän, olisi se löytynyt ylärivin kellosta ja akun prosenttimäärästä.

Neliöllisen erotuksen tulos saadaan osakuvan ja alkuperäisen kuvan vertailukohdan erotuksen neliöstä. Tällä menetelmällä samanlaisuus tulosmatriisista tulkitaan matriisin minimikohdista, sillä mitä lähempänä osakuvan ja vertailualueen pikseliarvot ovat toisiaan, sitä lähempänä nollaa niiden erotus on. Tämä eroaa muista menetelmistä, joissa yhdenkaltaisuus näkyy korkeana positiivisena arvona. [4] Esimerkiksi yhden vertailukohdan tulos R saadaan kaavasta (1) tai (2) riippuen siitä maskeerataanko kuvaa ensin,

$$R(x,y) = \sum_{x',y'} \left(T(x',y') - I(x+x',y+y')\right)^2 \quad (1)$$

$$R(x,y) = \sum_{x',y'} ((T(x',y') - I(x+x',y+y')) \cdot M(x',y'))^2 \quad (2)$$

, jossa T on Template eli osakuva, I on Image eli alkuperäinen kuva ja M on Mask eli kuvan maski. Molemmissa kaavoissa summataan osakuvan kokoisen alueen jokaisen pikselin neliöllisen erotuksen tulos, ja summa tallennetaan x:n ja y:n mukaiseen kohtaan tulosmatriisissa. [5]

Korrelaatiokerroinmenetelmä perustuu korrelaatiokertoimeen, joka on laajasti käytetty todennäköisyyslaskennan käsite, jolla mitataan kahden satunnaismuuttujan lineaarisen riippuvuuden voimakkuutta. Korrelaatiokerroin saa arvoja väliltä [-1, 1], jossa 1 merkitsee vahvaa lineaarista riippuvuutta ja -1 merkitsee vahvaa käänteistä riippuvuutta. Mitä lähempänä 0:aa korrelaatiokertoimen arvo on, sitä vähemmän riippuvaisia ovat muuttujat. Tämän ansiosta kyseinen menetelmä on vikasietoisempi ja kestää kohinaa ja häiriötä. Sitä voidaan soveltaa eri vektoreiden ja siitä soveltaen matriisien samankaltaisuuden vertailuun. OpenCV:ssä korrelaatiokerroinmenetelmän tulos yksittäiselle vertailupisteelle saadaan kaavalla,

$$R(x,y) = \sum_{x',y'}\left(T'(x',y') \cdot I'(x+x',y+y')\right) \tag{3}$$

, jossa

$$T'(x',y') = T(x',y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} T(x'',y'') \tag{4}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} I(x+x'',y+y'') \tag{5}$$

, joissa on samat merkinnät kuin kaavoissa (1) ja (2). Kaavat (4) ja (5) kuvaavat ristikorrelaation kaavan normalisointia pisteiden kovarianssilla. [5, 6]

Ristikorrelaatio on menetelmänä hyvin epästabiili ja sopimaton osakuvan etsintään, jos sitä ei normalisoida, joten sen yleisin muoto nykypäivän sovelluksissa on normalisoitu. Ei-normalisoitu ristikorrelaatio tuottaa vääriä tuloksia osakuvan etsinnässä, koska osakuvan maksimiarvo voi olla erisuuri alkuperäisen maksimiarvon kanssa, jolloin alkuperäisen kuvan kirkkaimmat pikselit tekevät piikkejä tulosmatriisiin, jotka tulkitaan väärin tuloksiksi. Ongelma saadaan ratkaistua normalisoinnilla, jolloin kahta kuvaa, joiden arvojen vaihteluväli on erisuuri, voidaan vertailla. [4] Normalisoidulla ristikorrelaatiolla laskettu tulos R saadaan kaavoista,

$$R(x,y) = \frac{\sum_{x',y'}\left(T(x',y') \cdot I(x+x',y+y')\right)}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{6}$$

$$R(x,y) = \frac{\sum_{x\prime,y\prime}\left(\left(T(x',y') \cdot I(x+x',y+y')\right) \cdot M(x\prime,y\prime)^2\right)}{\sqrt{\sum_{x',y'}(T(x',y') \cdot M(x\prime,y\prime))^2 \cdot \sum_{x',y'}(I(x+x',y+y') \cdot M(x\prime,y\prime))^2}} \tag{7}$$

, jossa merkinnät ovat yhtenäiset kaavojen (1) ja (2) kanssa. Molemmissa kaavoissa osoittajassa on ristikorrelaation kaava ja nimittäjässä normalisoiva termi. [5]

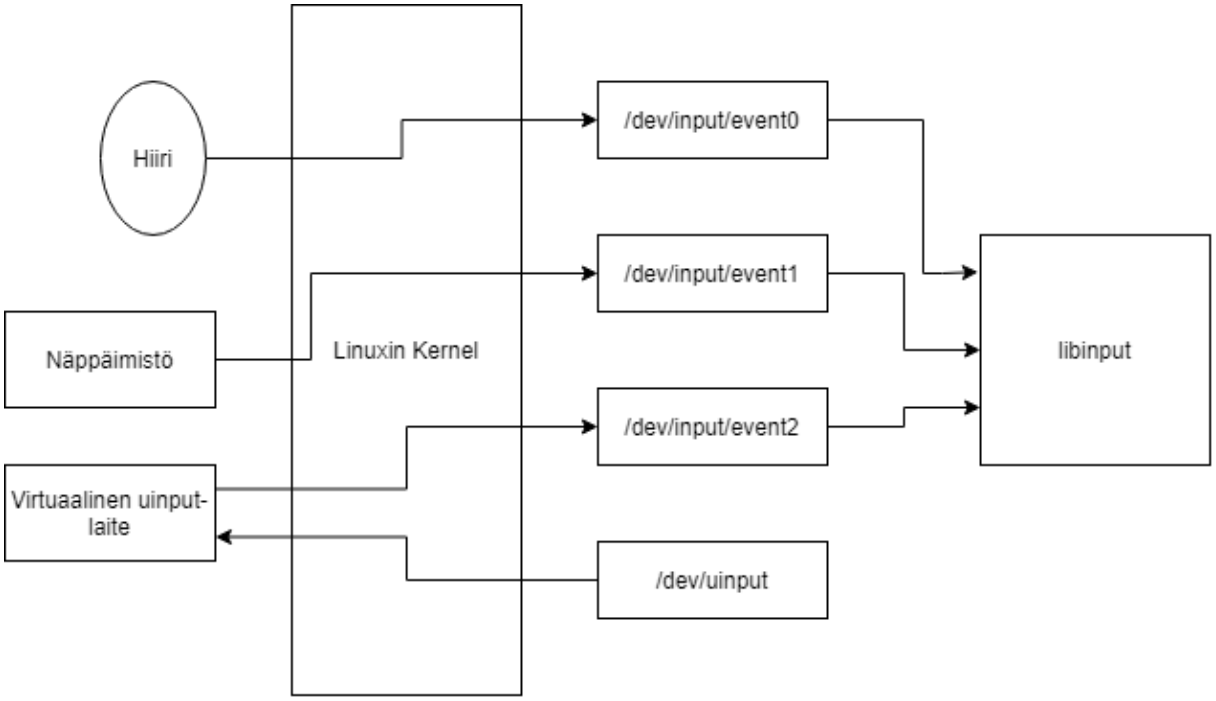
3. TEKNINEN OSA

Työn teknisessä osassa toteutettiin käytännön sovellus, jossa sovelletaan konenäköä sulautetun järjestelmän graafisen käyttöliittymän testaamiseen. Tavoitteena oli luoda työkalut, joiden avulla olisi helppo tuottaa useita erialisia testejä. Työssä käytettiin OpenCV kirjastoa python-ohjelmointikielellä, jonka avulla tarvittavat konenäön menetelmät saatiin yksinkertaisesti käyttöön. Valmiit työkalut liitettiin testiautomaatio kehykseen, jossa testien kehittäjät voivat soveltaa ja luoda testejä niiden avulla.

3.1. Virtuaalisen näppäimistön toteutus Uinput-moduulilla

Ohjausosa on erillinen ohjelmisto, joka asennetaan sulautetulle järjestelmälle. Tällä ohjelmistolla pystytään ohjaamaan laitetta painamalla virtuaalisia painikkeita. Ohjausosa ohjelmoitiin C-kielellä ja siinnä hyödynnettiin Linux kernelin Uinput moduulia. Uinput moduulilla pystytään luomaan virtuaalinen näppäin, jolla voidaan lähettää Linuxin kernelille näppäinkomentoja, jotka tulkitaan normaaleiksi näppäimen painalluksiksi. Näppäinkoodit järjestelmän näppäimille luettiin laitteelta hexdump-ohjelmalla lukemalla tapahtumapuskuria samalla kun painaa jotain näppäintä. Virtuaalinen näppäimistö luotiin Linuxin kernelin Uinput moduulilla, jolla pystyttiin luomaan virtuaalinen syötelaite, jonka sisältämät näppäimet voidaan valita alustuksessa.

Kuvassa 2 näkyy Uinputin paikka suhteessa muihin syötelaitteisiin. Virtuaalisen laitteen alustus tapahtuu kernelin sisäisesti, mutta laite muodostuu käyttäjätilaan, jolloin sen lähettämät tapahtumat näkyvät kernelille tavallisen syötelaitteen mukaisesti. Koska kernelin näkökulmasta virtuaalinen syötelaite on syötelaite muiden joukossa, sen tapahtumat menevät täysin samaa reittiä. Tämä mahdollistaa syötteiden luomisen järjestelmällä ajettavassa koodissa.



Kuva 2. Uinputin paikka linuxin kernelissä.

Ohjausosan toteutustavasta johtuen jokainen tapahtuma kestää noin kaksi sekuntia. Toteutus tehtiin Uinput moduulin dokumentaation pohjalta. Käytännössä kernelin

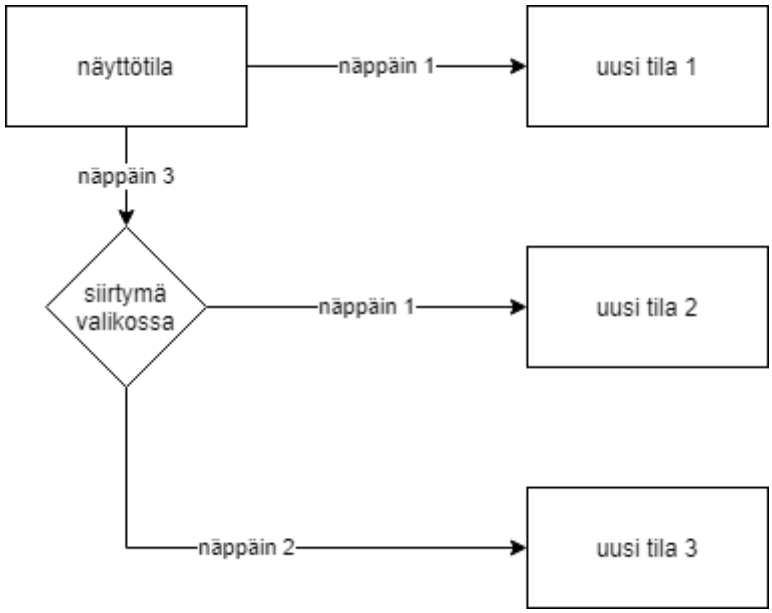
dokumentaation koodi parametrisoitiin ja sille toteutettiin kyvykkyys lukea komentoriviltä parametrit. Ohjausosaa käytetään järjestelmän komentoriviltä kutsumalla ohjelmaa ja antamalla sille parametrinä halutun näppäimen koodi, jolloin ohjelma luo virtuaalisen syötelaitteen ja rekisteröi sille halutun näppäimen. Näppäintä painetaan ja sen jälkeen se tuhotaan. Virtuaalisen syötelaitteen luomisen jälkeen ja ennen sen tuhoamista on odotettava noin sekunnin verran, jotta kernelin käyttäjätila huomaa laitteen ja kerkeää rekisteröidä tapahtuman ennen laitteen tuhoutumista. [3]

3.2. Näyttötilojen tilakartta

Näyttötilojen siirtymät kirjoitettiin talteen JSON-muotoiseen tiedostoon. Tilat sijoitettiin listaan, ja niille annettiin avainarvoiksi esimerkkikuvan polku, kuvamaskin polku, tilasiirtymien näppäinpainallusten kartta. Tilasiirtymät kuvattiin antamalla avaimeksi näppäimen tunniste, jonka arvona oli joko seuraavan tilan nimi tai sanakirja näppäimiä, joille annettiin arvot samalla tavalla. Tällä kartoitustavalla pystyttiin kattamaan tilat, joissa oli lista vaihtoehtoja, joihin pääsy vaati useamman näppäinpainalluksen. Esimerkiksi valikot ovat tällaisia tiloja, sillä liikuttaessa valikossa samalla näppäimen painalluksella voi olla eri tulos. Tällä lähestymistavalla eliminoitiin ylimääräiset tilamäärittelyt, jotka olisivat muodostuneet jokaisesta valikon vaihtoehdosta. Kuvassa 3 rakennettu esimerkki jostain näyttötilasta JSON muotoisena ja sitä on havainnollistettu kuvassa 4 vuokaavion muodossa. [Liite 1]

```
{
"näyttö_tila":{
    "esmierkkikuvan_polku": "path/to/file",
    "maskin_polku": "path/to/file",
    "näppäin_vaihtoehdot": {
        "näppäin_1": "uusi_tila_1",
        "näppäin_3": {
            "näppäin_1": "uusi_tila_2",
            "näppäin_2": "uusi_tila_3",
            }
        }
    }
}
```

Kuva 3. Esimerkki tilakartan rakenteesta.



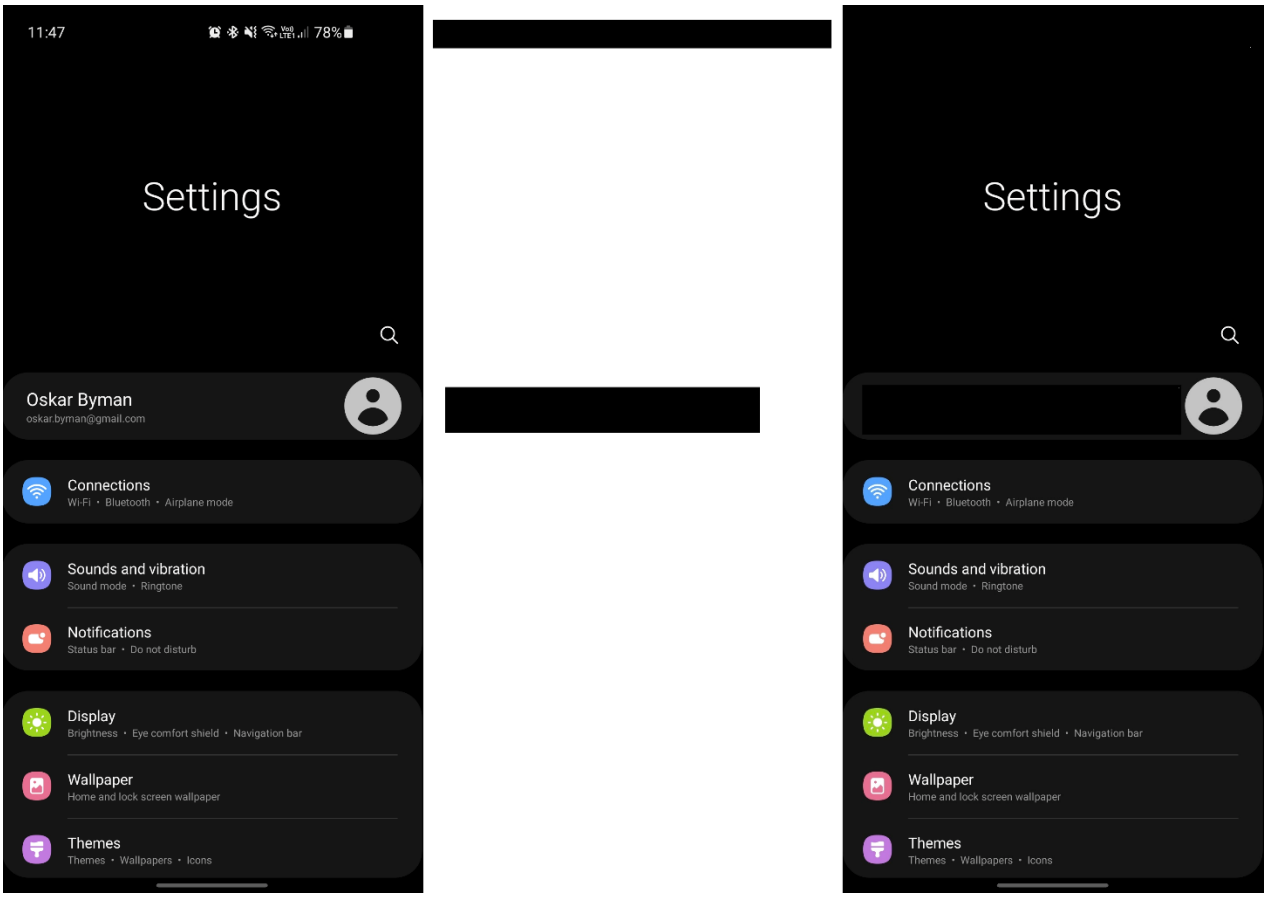
Kuva 4. Tilakartta visuaalisena kaaviona.

Tilakartta toimii rinnakkain järjestelmän kanssa, kun näppäin sarja syötetään laitteelle, se syötetään myös tilakarttaa käsittelevälle metodille. Siirryttäessä tietyllä näppäinpainallusten sarjalla johonkin näyttötilaan, pystytään kartan avulla päättelemään missä tilassa järjestelmän tulisi olla. Tilakartasta saadaan silloin haluttu tila ja sen perusteella oikea maski ja esimerkkikuva. Tällöin voidaan tarkastella, onko järjestelmän tilasiirtymät yhdenmukaisia tilakartan kanssa.

3.3. Kuvakaappaukset ja maskeeraus

Kuvakaappaukset saatiin kopioimalla Linux ohjelmiston näyttöpuskurista pikselien RGB tiedot erilliseen tiedostoon, jonka jälkeen käytettiin ffmpeg ohjelmistoa kääntämään kuva raa'asta muodosta PNG tiedostomuotoon. Linux käyttöjärjestelmän näyttöpuskuri löytyy oletusarvoisesti polusta /dev/fb0. Kuvat kopioitiin järjestelmältä erilliselle laitteelle, jossa ne muutettiin käsiteltävään PNG muotoon. Lopullisissa testeissä kuvat haetaan samalla tavalla, jolloin ulkoisten häiriöiden vaikutus minimoidaan, kun toteutusta verrataan esimerkiksi erillisellä kameralla kuvattuun käyttöliittymään. Vaihtoehtoinen ohjelma kuvan muuntamiselle on ImageMagickin convert komento.

Kuvan maskeeraus on maskin ja alkuperäisen kuvan AND-bittioperaatio. Maski on mustavalkoinen kuva, jossa valkoiset pikselit sisältävät arvon yksi ja mustat pikeslit arvon nolla. Tällöin maskeerauksen jälkeen kuvaan jää ainoastaan valkoisten pikseleiden kohdilla olevat alkuperäisen kuvan pikselit. Esimerkki vaiheista kuvassa 5. Maskit luotiin käyttämällä OpenCV kirjastolle valmiiksi Bittiumin sisäisesti kehitettyä apuohjelmaa, jolla pystyttiin aukaisemaan kuva ja piirtämään siihen suorakulmioilla maskeerattavat alueet. Maskin tulee olla samankokoinen ulottuvuuksiltaan kuin alkuperäisen kuvan, jotta maskin ja kuvan AND-operaatio onnistuu oikein. Kuvista maskeerattiin kaikki muuttuvat osat, kuten kellonaika. Tällöin kuvia voitiin verrata maskeerauksen jälkeen nollatoleranssilla, koska niiden varmasti muuttuvat osat pystyttiin poistamaan vertailusta. [Liite 2]



Kuva 5. Kuvakaappauksen maskeerauksen kolme vaihetta: Kuvakaappaus, maskin luonti ja maskeeraus.

3.4. Järjestelmää kuvaava luokka

Laitteen hallintaa koodissa helpottaa, jos laite kuvataan objektina koodissa, koska sille voidaan toteuttaa erinäisiä metodeja. Luokkaan toteutettiin metodit ohjausosan käyttöön, graafisen käyttöliittymän uudelleenkäynnistämiseen ja näyttökuvan kaappauksen metodi. Käytännössä luokkaan toteutettiin koodissa käytettävät menetelmät ohjata ja käyttää laitteen eri ominaisuuksia. Liitteessä 3 on kuvattu python kielisen luokan rakenne, mutta metodien toteutus on jätetty tietosuojan vuoksi pois. [Liite 3]

Luokkamuotoinen toteutus tehtiin, koska muut automaation osat käyttivät luokkamuotoista rakennetta. Olio-ohjelmoinnin periaatteet mahdollistavat koodin uudelleenkäytön useassa paikassa, jolloin toteutusta voidaan käyttää yhtä aikaa usealla laitteella.

3.5. Kuvien tulkinta

Lopullisissa testeissä on tarve tulkita kuvasta järjestelmän tila ja löytää siitä jokin osakuva. Järjestelmän tilaa tulkitaan ottamalla järjestelmän näytöstä kuvakaappaus, joka maskeerataan ja verrataan maskeerattuun esimerkkikuvaan. Kuvat ovat matriisimuodossa ohjelmassa, jossa jokainen elementti kuvastaa yhtä pikseliä laitteen näytöllä. Vertailu suoritetaan matriisien kesken tarkistamalla niiden yhtäsuuruus ja samanlaisuus. OpenCV:ssä kuvia käsitellään

kuvaobjekteina, joita pystytään vertailemaan kuin mitä tahansa muuttujia. Työssä tarkastettiin muuttujien yhtäsuuruus ja molempien matriisien jokainen elementti verrattiin toisiinsa XOR-funktiolla.

Osakuvien etsintään käytettiin OpenCV:n Template Matching- metodia korrelaatiokerroinmenetelmällä. Template Matching tarkoittaa osakuvan etsintää jostain suuremmasta kuvasta. Metodille annetaan parametreinä haluttu menetelmä, kynnysarvo ja kuvat. Menetelmäksi valittiin korrelaatiokerroin, jossa verrataan osakuvaa kokonaiskuvaan liu'uttamalla sitä kokonaiskuvan päällä ja vertailemalla joka kohdassa kuvia toisiinsa. Jokaisessa kohdassa lasketaan osakuvan ja alkuperäisen kuvan korrelaatiokerroin ja se merkataan tulosmatriisiin. Tulosmatriisin korkeiden arvojen ja kynnysarvon perusteella valitaan kaikki lähimpänä osakuvaa olevat kohdat alkuperäisestä kuvasta. Kuvassa 1 havainnollistetaan osakuvan etsinnän vaiheet. [Liite 3]

3.6. Integrointi ja testien kehitys

Integraatio testikehykseen vaatii järjestelmää kuvaavan luokan, jolle toteutetaan metodeja, jotka kuvastavat fyysisen järjestelmän toimintoja, kuten painikkeen painallusta. Luokkaan toteutetaan sellaisia metodeja mitä laitteella käyttäjä pystyisi tekemään. Testejä varten on luotava myös kirjasto, jossa luodaan korkeamman abstraktiotason metodeja, kuten painikesarjan painelu. Testikirjaston ideana on käyttää järjestelmää kuvaavan luokan yksinkertaisia metodeja ja tehdä niistä järkevän tapaisia sekvenssejä. Esimerkkinä luokkaan toteutettu graafisen käyttöliittymän uudellenkäynnistys ja näppäimenpainallus, joiden avulla testikirjastoon toteutettiin metodi, joka alustaa käyttöliittymän käynnistämällä sen uudelleen ja näppäilemällä tarvittavat tiedot.

Testikirjaston pohjalta voidaan luoda testisekvenssejä kirjoittamalla kirjaston metodeja kronologiseen suoritusjärjestykseen. Tuotetut esimerkit olivat, tunnetun näppäinsarjan syöttö ja tilan tarkistus, satunnaisen näppäinsarjan syöttö ja osakuvan etsintä. Kaikissa testisekvensseissä alustettiin järjestelmän käyttöliittymä ja sen jälkeen suoritettiin testi. Testien perusteella korjattiin useita virheitä työssä tuotetusta koodista. [Liite 4]

3.7. Tulokset

Työn testaaminen suoritettiin ajamalla automaatiotestejä työn aikana tuotetuilla testeillä. Tärkeimpänä testinä pidettiin satunnaisten näppäinsarjojen suorittamista, jolloin voitaisiin löytää satunnaisesti rajatapauksia tuotetusta kirjastosta. Testien perusteella pystyttiin tunnistamaan laitteen tila sisäisen tilakartan mukaisesti ja onnistuttiin etsimään osakuva jostain laitteen tilasta. Työn avulla päästiin tarkastelemaan laitteen tilojen stabiilisuutta ilman käyttäjän toimenpiteitä.

Testien aikana ilmeni ohjausosan aiheuttama hitaus. Koska jokainen ohjausosan suoritus kesti noin kaksi sekuntia, Järjestelmän alustuksessa meni kymmeniä sekunteja, kun lasketaan mukaan graafisen käyttöliittymän uudelleenkäynnistyksestä aiheutunut viive ja alustustoimenpiteiden vaatimat näppäinpainallukset. Tämä rajoittaa ajallisesti yhdellä laitteella suoritettavien testien määrää. Yön yli 16 tunnin mittainen automaattinen testaus rajoittuisi noin 1500 testisekvenssiin, jos sekvenssit olisivat keskimäärin 38,4 sekuntia pitkiä. Todellinen määrä yön aikana ajettavia testisekvenssejä olisi reilusti pienempi, johtuen automaatiokehyksen ominaisista toimintatavoista ja testiympäristön alustustoimenpiteistä. Testatussa 56 tilaisessa käyttöliittymässä erilaisia sekvenssejä on paljon, kun ottaa huomioon tilantarkistustestit ja

osakuvien tarkastelutestit, mutta ohjausosa rajoittaa sitä. Rajoituksesta johtuen on siis valittava kriittiset testisekvenssit, joita voidaan ajaa useammin. Ohjausosan vaikutus ei rajoitu ainoastaan graafisen käyttöliittymän testaukseen sillä, jotkin eri toiminnallisuuden testit vaativat joitain tapahtumia käyttöliittymältä.

Työssä käytetty toteutusmenetelmä on kankea ylläpitää, sillä rakennettua JSON muotoista tilakarttaa joudutaan päivittämään käsin. Tilakartta todettiin testauksessa toimivaksi, mutta sen päivitys vaatii jatkuvaa synkronointia kehittäjien ja testaajien välillä.

4. POHDINTA

Työn teknisessä osassa tuotettu konenäköä käyttävä testiautomaatio kirjasto pääsi alussa asetettuihin tavoitteisiin, mutta tilaa jatkokehitykselle jäi.

Sulautetun järjestelmän ohjausosan toteutus tehtiin Uinput moduulin esimerkin mukaisesti, joka johti kahden sekunnin viiveeseen jokaisella näppäimen painalluksella. Viive aiheutuu siitä, kun virtuaalinen näppäin alustetaan ja tuhotaan jokaisella käyttökerralla. Luomisen ja painamisen välissä on oltava noin sekunnin viive, jotta Linuxin kernel ehtii huomata näppäimen ennen kuin sille lähetetään tapahtumia. Toteutusta on tarkoitus korjata tulevaisuudessa tekemällä ohjausosasta taustalla pyörivä ohjelma, jonka kanssa kommunikointi tapahtuisi jonkinlaisen rajapinnan kautta. Tällä tavalla ohjausosa voitaisiin käynnistää ja alustaa aina testit sekvenssin alussa ja näppäinten painallukset olisivat huomattavasti nopeampia.

Tilakartan kankeus on toinen suuri kehityskohde. Ylläpitokuorman vähentäminen tilakartan ratkaisulla olisi tärkeä muutos. Tälle mahdollinen ratkaisu on OCR, joka mahdollistaisi tekstin tulkinnan näytöstä, jolloin ne tilat, joissa on otsikko näkyvillä, voitaisiin tunnistaa suoraan ilman tilakartan seurantaa. Rajatapauksia syntyisi niistä tiloista, joissa tekstiä ei ole ollenkaan, mutta ne voitaisiin tunnistaa edelleen jo toteutetulla tavalla. toteutuksen tehokkuus ei välttämättä muuttuisi, mutta sen ylläpitotyö olisi kevyempää.

5. YHTEENVETO

Työssä toteutettiin konenäköä soveltava kirjasto, jonka avulla sulautetun järjestelmän graafisen käyttöliittymän testausta voitiin automatisoida. Työ toteutettiin Bittium Oyj:lle projektin tarpeeseen.

Konenäön pitkä historia ulottuu 50 vuotta taaksepäin, jonka aikana on edetty eri osa-alueilla paljon. Osakuvan etsintä, englanniksi Template Matching, perustuu tilasto- ja todennäköisyyslaskennan menetelmiin. OpenCV:n TM metodi voi käyttää monia eri menetelmiä, joista esiteltiin Neliöllinen erotus, Korrelaatiokerroin ja Normalisoitu ristikorrelaatio.

Toteutusosassa kehitettiin automatisoidut testit, joilla järjestelmän käyttöliittymää voidaan toistuvasti testata. Testien kehitykseen tarvittiin useita osia, kuten ohjausosa, jolla luodaan laitteelle virtuaalinen syötelaite, jonka avulla voidaan ohjata laitetta. Ohjausosa toteutettiin parametrisoimalla Linux kernelin Uinput moduulin esimerkkiratkaisu. Ohjausosan lisäksi luotiin laitetta kuvaava luokka, joka sisälsi metodit laitteen käsittelyyn ja tilakartan eri näyttötilojen välille. Testejä varten tehtiin testikirjasto, jossa luotiin yksinkertaisia sekvenssejä käyttäen luokkaan luotuja metodeja. Tähän kuului laitteen tilan tarkistus ja laitteen näytöltä osakuvan etsintä teoriaosassa käsitellyllä korrelaatiokerroin menetelmällä.

Työn tuloksena oli toimiva ensimmäinen versio toteutuksesta. Siitä löytyi kehityskohteita, joille etsittiin jo alustavia ratkaisuja jatkokehitystä varten. Työ otettiin käyttöön ja sen pohjalta aloitettiin uusien testien kehitys.

6. LÄHDELUETTELO

[1] Szeliski R. (2010), Computer Vision: Algorithms and Applications, Springer Science & Business Media. 812 s.

[2] Lowe D. (Luettu 15.2.2021) The Computer Vision Industry URL: https://www.cs.ubc.ca/~lowe/vision.html

[3] Linuxin kernel documentation, Uinput module, Luettu 20.6.2021 https://www.kernel.org/doc/html/v4.12/input/uinput.html

[4] Hisham M. B., Shahrul N. Y., Raof R. A. A., Nazren A.B A., Wafi N. M. (2015) Template Matching Using Sum of Squared Difference and Normalized Cross Correlation. In: IEEE Student Conference on Research and Development (SCOReD), December 13. and 14., Kuala Lumpur, Malaysia

[5] OpenCV documentation Object detection https://docs.opencv.org/4.5.2/df/dfb/group__imgproc__object.html

[6] Napoli N, Barnes L, Premaratne K (2015) Correlation coefficient-based template matching: Accounting for uncertainty in selecting the winner. In: 18th International Conference on Information Fusion, July 6-9, Washington, DC

7. LIITTELUETTELO

Liite 1 ScreenLinkList.json

Liite 2 image_comparison.py

Liite 3 DeviceClass.py

Liite 4 template_matching.py

Liite 5 Test.robot

Liite 1 ScreenLinkList.json

```
{
"näyttö_tila" :{
   "esmierkkikuvan_polku": "path/to/file",
   "maskin_polku": "path/to/file",
   "näppäin_vaihtoehdot": {
      "näppäin_1": "uusi_tila_1",
      "näppäin_3": {
         "näppäin_1": "uusi_tila_2",
         "näppäin_2": "uusi_tila_3",
         }
      }
   }
}.
```

Liite 2 image_comparison.py

```
import cv2
import numpy as np

def are_images_identical(image, ref_image, mask):
    """
    Lukee kuvat ja maskin, jonka jälkeen tarkistelee kuvien yhdensuuruutta.

    Parametrit:
        image: verrokkikuva
        ref_image: referenssikuva
        mask: kuville sopiva maski
    Palautusarvo:
        boolean: totuusarvo, olivatko kuvat samankokoisa ja jokainen pikseli oli samanlainen
    """

    img = cv2.imread(image)
    ref_img = cv2.imread(ref_image)
    msk = cv2.imread(mask, 0)

    masked_img = cv2.bitwise_and(img, img, mask=mask)
    masked_ref_img = cv2.bitwise_and(ref_img, ref_img, mask=mask)

    return masked_img.shape == masked_ref_img.shape and not
np.bitwise_xor(masked_img, masked_ref_img).any()
```

Liite 3 DeviceClass.py

```
class DeviceGUI(Device):
    """
    Esimerkkijärjestelmän graafisen käyttöliittymän luokkarakenne,
        joka periytyy esimerkkijärjestelmän luokasta.
    Kommenteissa funktioiden selitykset.
    """


    def __init__(self):
        super().__init__()
        #  Alustetaan luokan ominaisuudet, kuten nänppäinkoodit ja tilakartta

    def push_button(self, keycode):
        #  Painetaan haluttua näppäintä annetun näppäinkoodin perusteella
        pass

    def restart_gui(self):
        #  Käynnistetään graafinen käyttöjärjestelmä uudestaan
        pass

    def capture_screen(self, output):
        #  Otetaan näytön kuva talteen ja tallennetaan annettuun paikkaan
        #  Tässä funktiossa voidaan myös muuntaa kuva jo haluttuun muotoon
        pass
```

Liite 4 template_matching.py

```
import cv2
import numpy as np

def match_template(image, template, output, method=cv2.TM_CCOEFF, threshold=0.99):
  """
  Suorittaa osakuvan etsinnän annetusta kuvasta halutulla menetelmällä.
  Merkitsee kynnysarvon ylittävät tulokset haluttuun polkuun sijoitettavaan kuvaan
  HUOM! koodi ei tuo TM_SQDIFF tai TM_SQDIFF_NORMED metodeja,
    koska tulosten tarkastelussa tallennetaan kynnysarvon ylittävät tulokset.


  Parametrit:
    image: tarkasteltava kuva
    template: etsittävä osakuva
    output: tuloskuvan polku
    method: käytetty template matching menetelmä
    threshold: tunnistettujen osakuvien kynnysarvo
  """
  img_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
  w, h = template.shape[::-1]
  result_matrix = cv2.matchTemplate(img_gray, template, method)
  locations = np.where(result_matrix >= threshold)
  for point in zip(*locations[::-1])
    cv2.rectangle(image, point, (point[0] + w, point[1] + h), (0, 0, 255), 2)
  cv2.imwrite(output, image)
```

Liite 5 Test.robot

```
*** Settings ***

Library  TestLibrary.py

*** Test Cases ***

Initialization
  Restart GUI
  Initialize GUI  Param 1  Param 2

Known button sequence
  Restart GUI
  Initialize GUI  Param 1  Param 2
  Push button sequence  Left  Right  Power  Power  Up
  Is screen expected

Random button sequence
  Restart GUI
  Initialize GUI  Param 1  Param 2
  Push random button sequence  9
  Is screen expected

Find icon in settings
  Restart GUI
  Initialize GUI  Param 1  Param 2
  Push button sequence  Left  Left  Left
  Look for template  ${path/to/template}
```